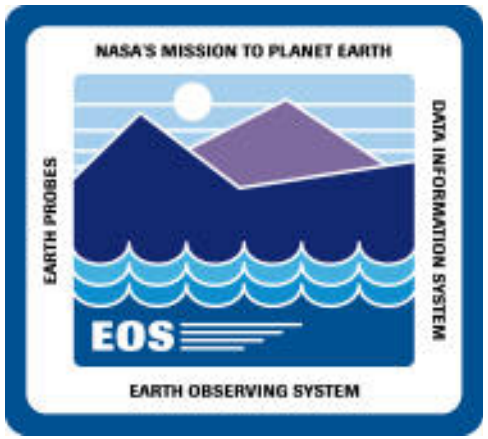


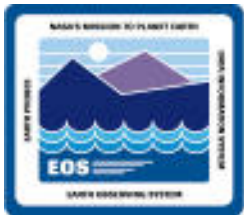
Process Framework Design Review

Shankar Rachakonda

vrachako@eos.hitc.com

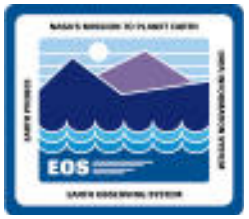
17 April 1996

Process Framework (PF) Overview



- **Driving Requirements**
- **PF Software Design**
- **PF Context**
- **OO Design Models**
- **Design Drilldown**

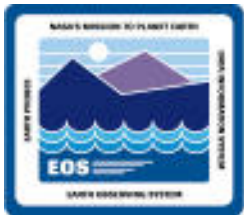
PF Driving Requirements



- General Functional Description
  - an extensible mechanism for ECS developed client and server applications to transparently include ECS infrastructure features
- Key Requirements
  - Encapsulate implementation details of ECS infrastructure services and remove the need for programmers to rewrite common initialization code
  - Ensure design and implementation consistency for all ECS Client and Server Applications
- New Release B Features
  - New functionality for SDPS and CSMS in Release B
  - Under study for FOS
  - Retrofitted into Release A development
- Evolutionary Features
  - Provides a basis by which future extensions to infrastructure mechanisms can be incorporated without adversely affecting the ECS developers

For more details please refer to 305-CD-028-002, Section 4.5.1

PF Software Design



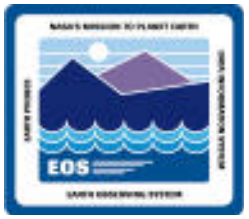
Three step approach

Step 1: Identify all the common capabilities needed in ECS client/server applications

Step 2: Classify ECS processes from a client/server perspective

Step 3: Allocate required capabilities at different levels of abstraction for each process type

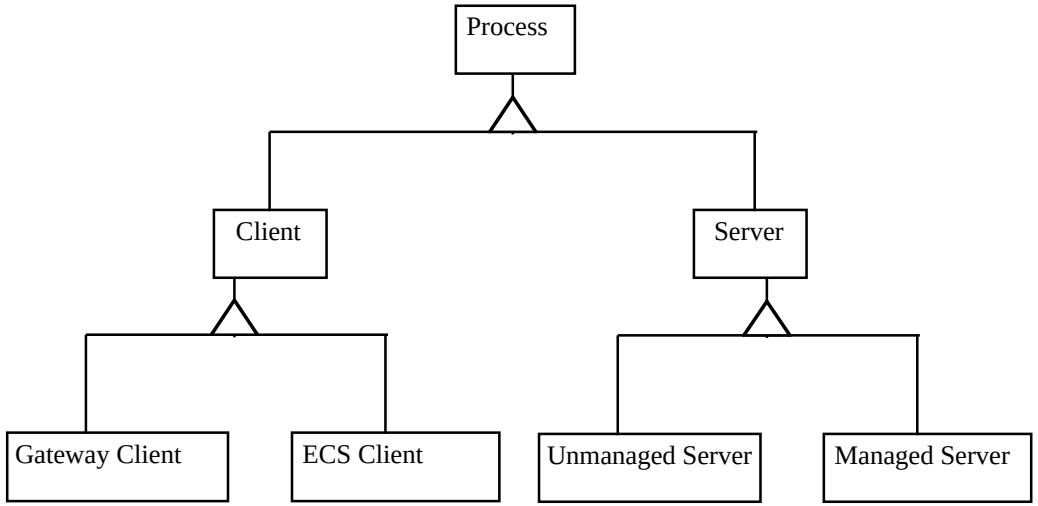
Common Capabilities Required by ECS Processes



- **Ability to initialize the process application and infrastructure in a consistent way and provide some basic process information**
- **Interface to Mode Management and Error-Event Handling**
- **Support for Life Cycle Services**
- **Interface to Asynchronous Message Passing, Server Request Framework and to common facilities such as batch FTP**
- **Encapsulation of OODCE Naming/Directory and Security parameters setup**

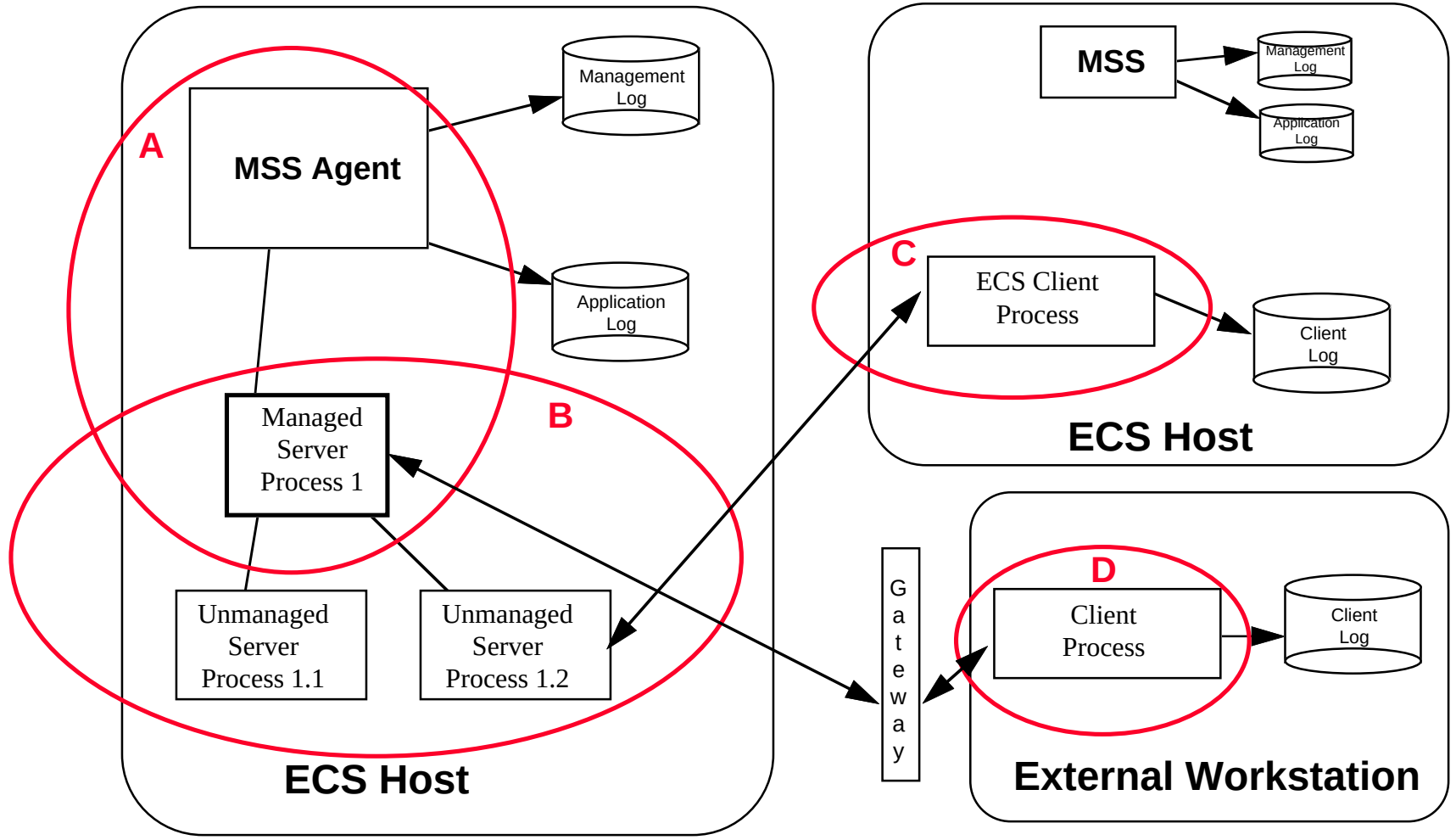
PF Software Design

ECS Process Classification



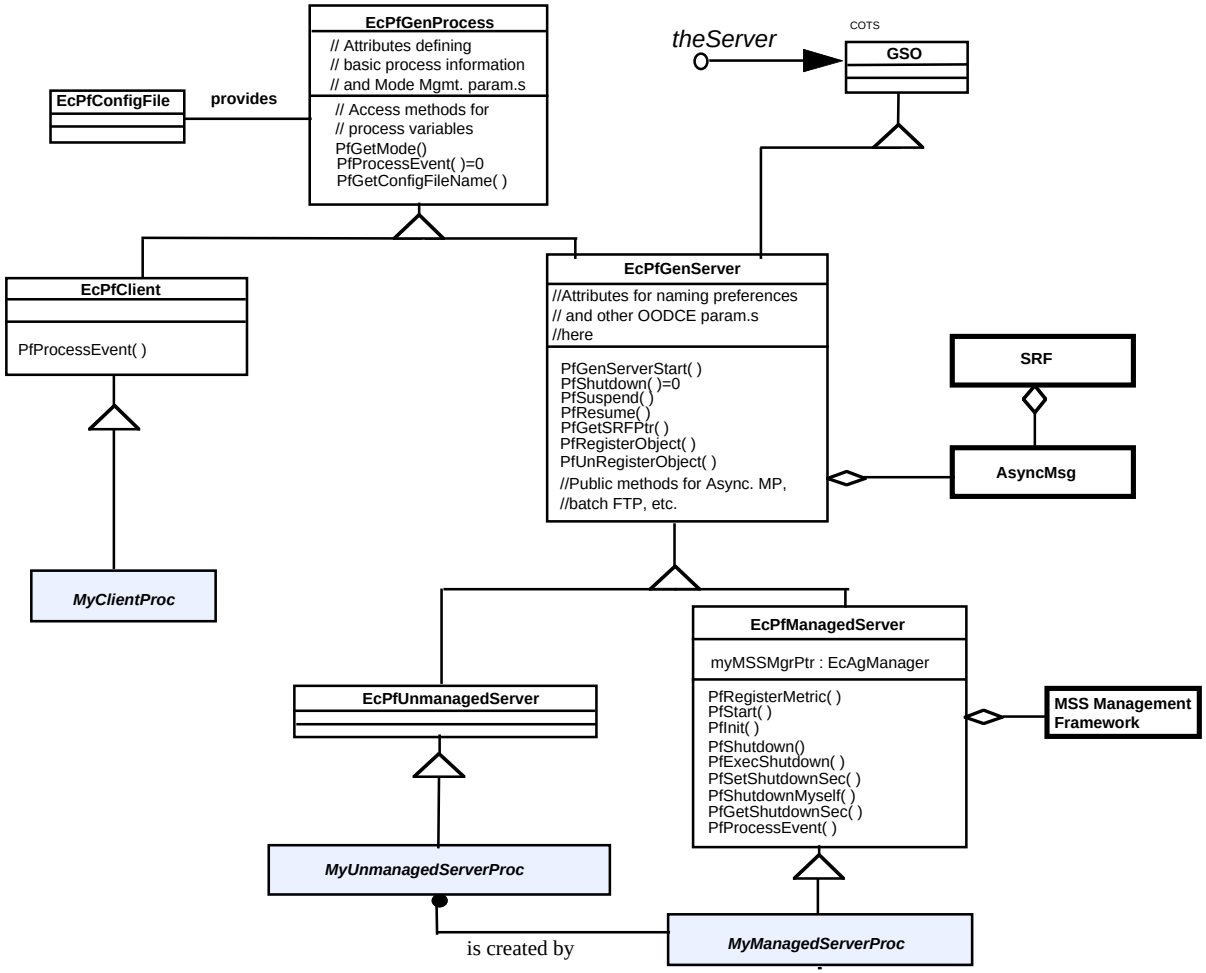
Document Reference:
305-CD-028-002,
Fig. 4.5.1.1-1

Operating Context of ECS Processes



Document Reference:305-CD-028-002,Fig. 4.5.1.2-1

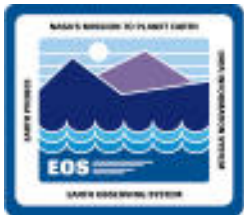
PF Software Design
Simplified Object Model



Class Descriptions

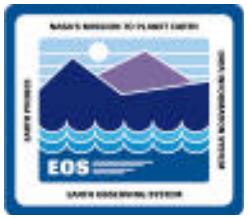
EcPfGenProcess	This class represents a generic process. It has all the common functionalities for all the processes. It is mainly a container of attributes needed by every process. It obtains attribute values from the configuration file or command line parameters.
EcPfGenServer	This class provides basic framework for server processes. It encapsulates DCEServer capabilities to provide OODCE naming/directory set up, and performs initialization of other services such as FTP, Message Passing, Security.
EcPfManagedServer	This class provides a basic framework for managed server processes. All ECS server applications are managed server processes. This class provides an interface with MSS subagent, and through which, ECS application can communicate with MSS. This class also provides interface with Event Handling, which allows consistent Error-Event logging for all ECS server application.

Class Descriptions



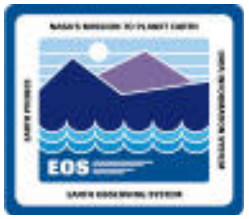
EcPfUnManagedServer	This class is a framework class for unmanaged server processes. Unmanaged server processes has all the functionalities of a managed server processes, except it is not under management of MSS.
EcPfClient	Defines the framework for client processes.
MyManagedServerProc	This class is implemented by programmers to inherit managed server process functionalities. This class inherits from EcPfManagedServer, and is application specific. Application programmers are required to provide implementation of application shutdown method of this class.
MyUnManagedServerProc	This class is implemented by programmers to inherit unmanaged server process functionality.
MyClientProc	This class is implemented by programmers to inherit client process functionality

Configuration File for PF



- Server process options are indicated in an orderly (parsable) fashion in a configuration file - like the .Xdefaults in Motif
- This file *can* be different for each instantiation of a server executable
- Code should not be recompiled to run with different options (ex: different modes)
- Several options have been identified
- Other options as identified by subsystems will be incorporated

Configuration file Syntax/ Options

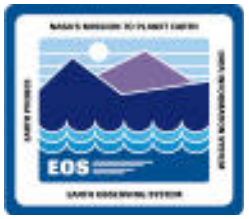



```
[PF]
FtpThreads = 5        // simultaneous ftp threads to do batch processing
ServerName = “myname”  // name of the server - one name per server
GroupName = “groupname”  // group name
ProfileName = “profilename” // profile name
Protocol = “tcp”                  // underlying transport protocol
Site = “ ”                        // site
Async = “True”                    // need asynchronous messaging ?
AsyncPstFile = “filename”         // async message persistence filename
KeyFile = “keyfilename”           // server identity (security) keytab file
HostPolicy = “one”                // one server per host
```

Application specific information can also be included

A template of the configuration file will be provided in a central place so application programmer can start with it (copy and provide values)

Configuration File Syntax Options



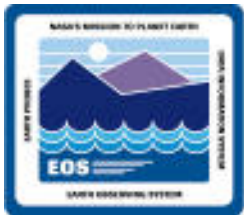
Precedence for setting options
(Listed below in the order of precedence)

1. **Access functions provided by the framework**

2. **Command line arguments**

3. **Configuration File entries**

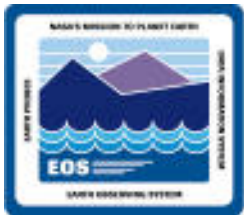
Object Model



The following object model will be reviewed:

Model Name	Document Reference	Section
PF Object Model	305-CD-028-002	4.5.1.3

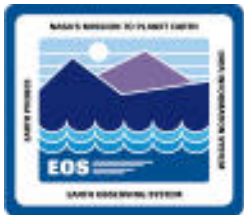
Dynamic Model



The following event traces will be reviewed:

Event Trace Name	Document Reference	Section
Application Start Up	305-CD-028-002	4.5.1.5.1
Application Shutdown	305-CD-028-002	4.5.1.5.2
Event Logging	305-CD-028-002	4.5.1.5.4

Event Traces



Application Start Up

Scenario

- This scenario describes the start up of a generic server application.

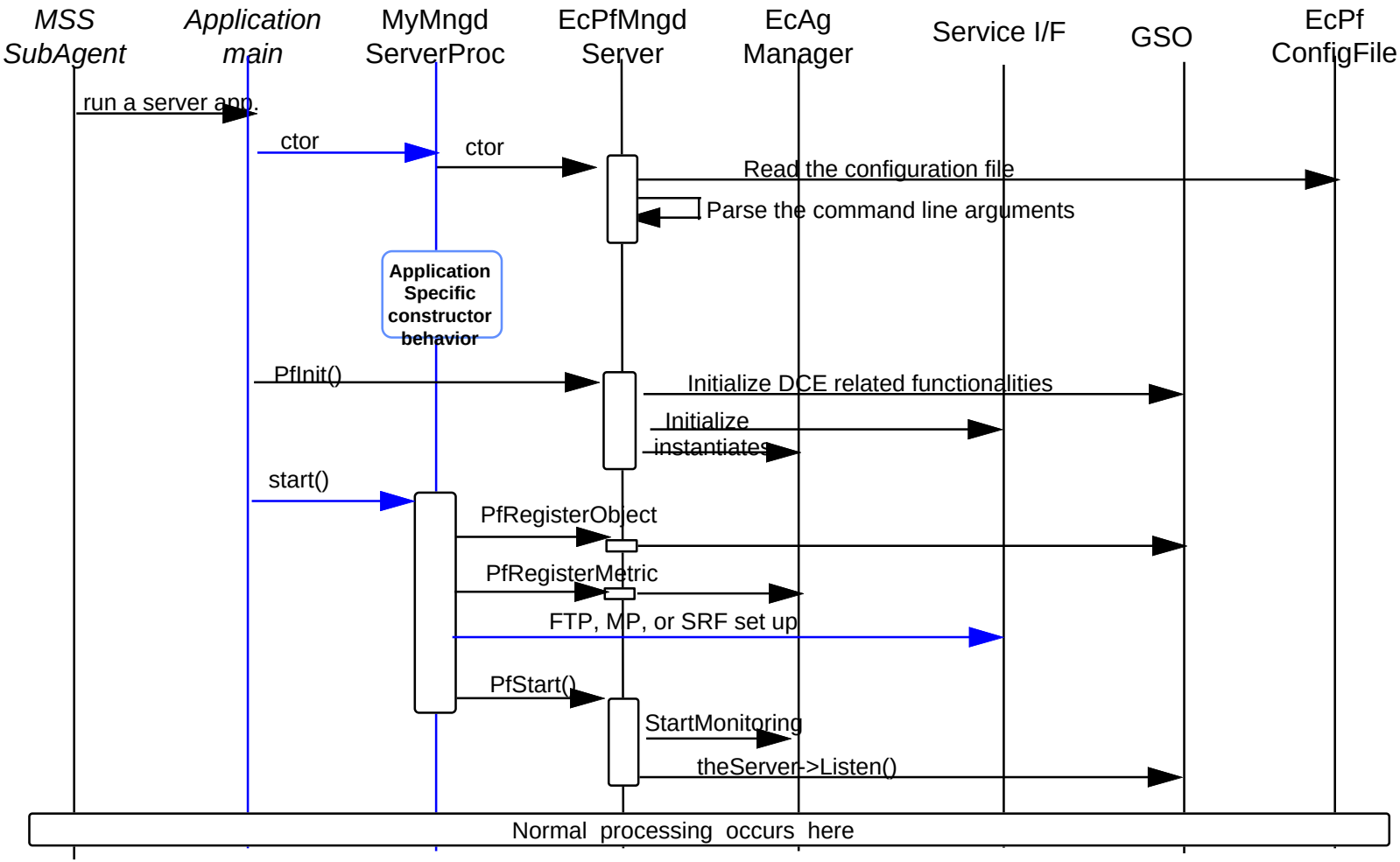
Functional Description

- MSS sub-agent fires the script for starting the ECS server application.
- Server application main instantiates a MyManagedServerProc object, and reads from a configuration file and command line arguments to set attributes needed for the process execution.
- Application main calls MyManagedServerProc->PfInit() to perform all DCE related initialization (and optionally SRF, message passing, and FTP initialization) for the process. EcAgManager object is created at this time.
- Application performs normal server setup such as creating server manager objects, registering objects with GSO, and registering metrics with MSS.
- Application main calls MyManagedServerProc->PfStart() to start MSS monitoring, and listen to client requests.

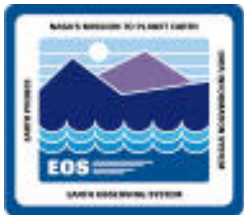
Assumptions/ Preconditions

- The application is available on an ECS host.

Startup Scenario



Event Traces



Application Shutdown

Scenario

- Shutdown of a generic server application at the request of MSS

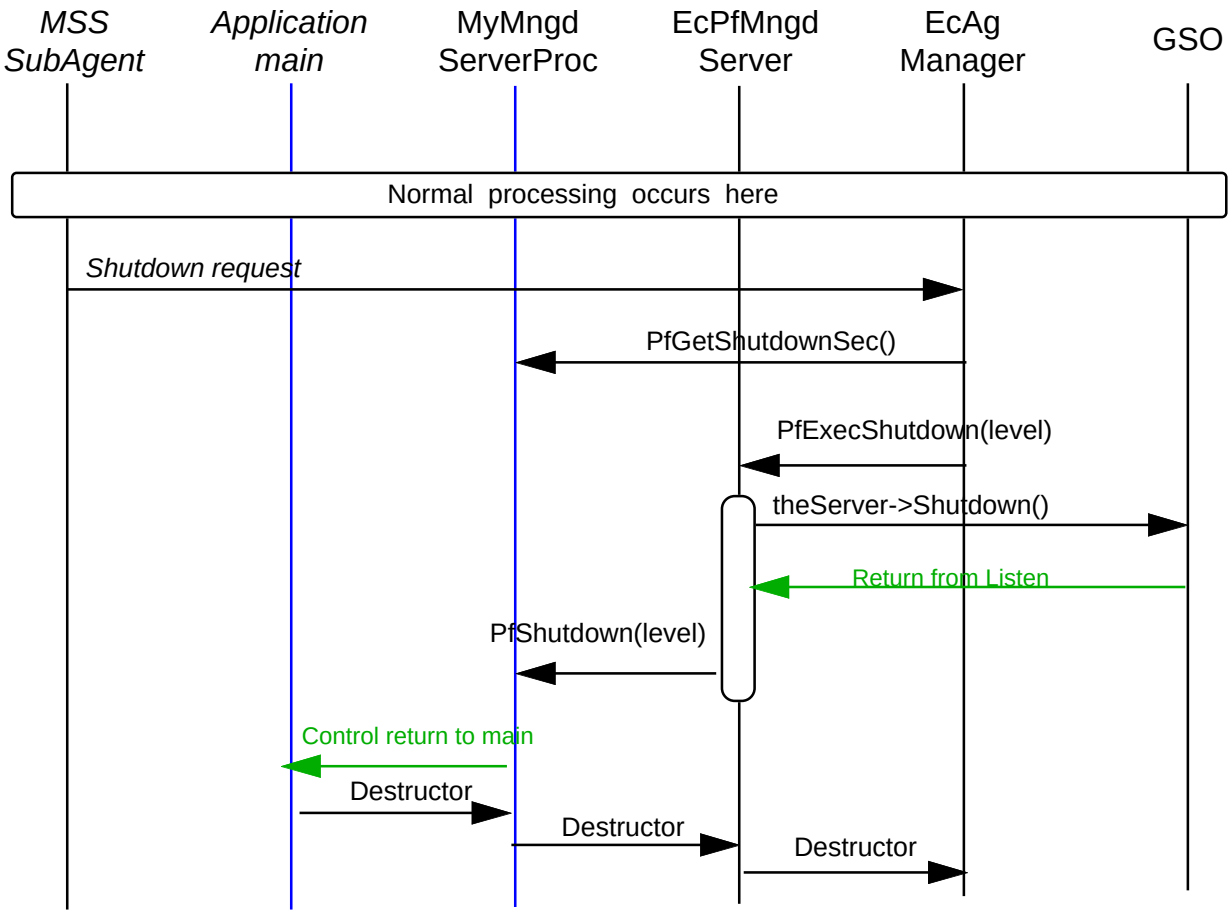
Functional Description

- MSS sub-agent, EcAgManager, calls PfGetShutdownSec() to obtain the number of seconds the application needs to perform shutdown.
- After the shutdown seconds is obtained, the MSS sub-agent then calls PfExecShutdown() to shutdown the server process.
- PfExecShutdown() will first call theServer->Shutdown() to stop accepting client requests, and return from listen.
- The application specific shutdown method (PfShutdown()) is called to perform application specific shutdown. If the application does not shutdown within the indicated time frame, the operator will be given an opportunity to kill the application or let it continue to operate.

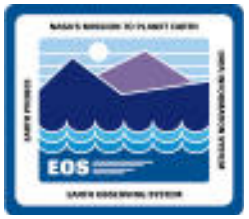
Assumptions/ Preconditions

- The application process is running.

Application Shutdown



Event Traces



Event Logging

Scenario

- For logging errors/events

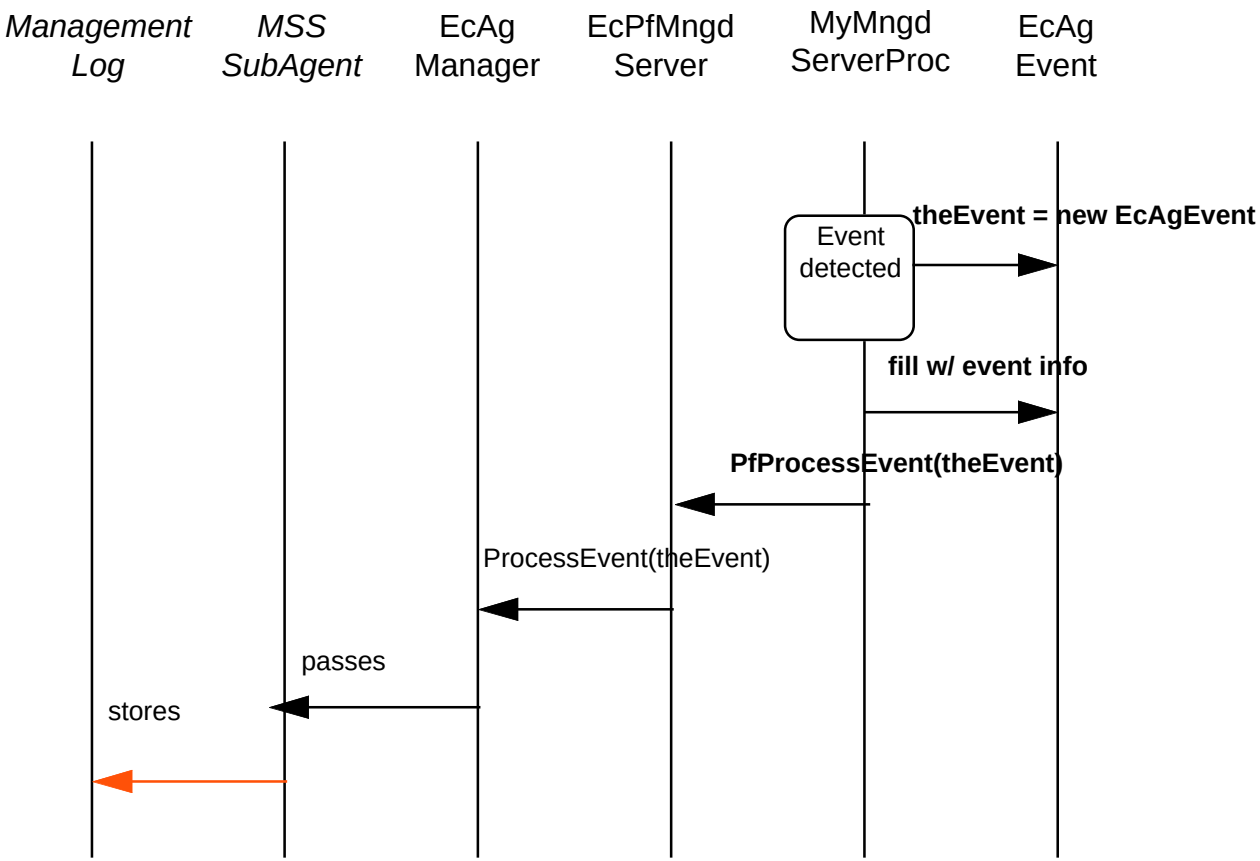
Functional Description

- On detecting an event the application creates an instance of the EcAgEvent class and uses it to describe the event.
- It then passes the event object, along with the log type, to the EcAgManager to be logged using the method ProcessEvent.

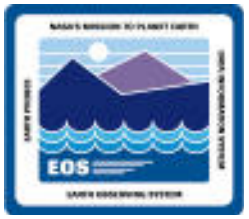
Assumptions/ Preconditions

- Logging has been initialized

Event Logging



ECS Process Framework



Development Recipe

Step 1: Develop a configuration file

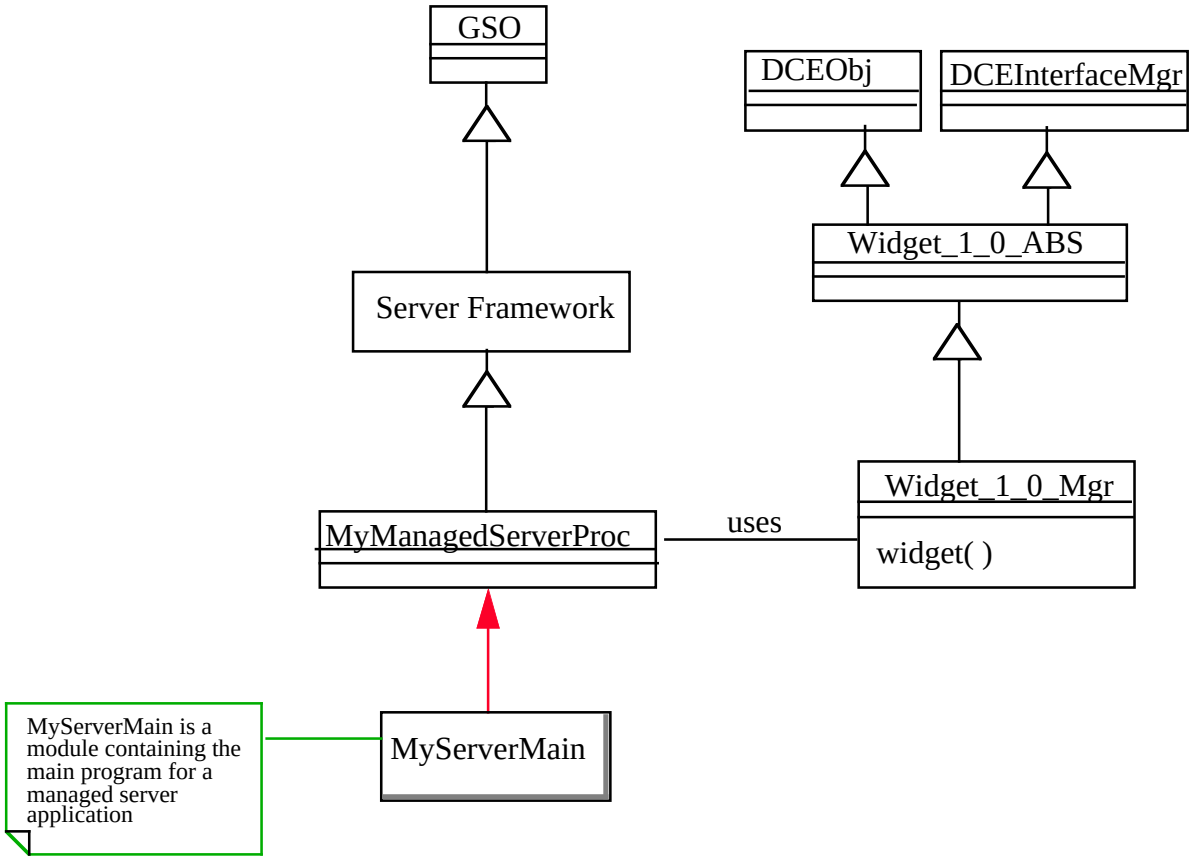
Step 2: Develop Distributed Objects

Step 3: Develop the appropriate derived class

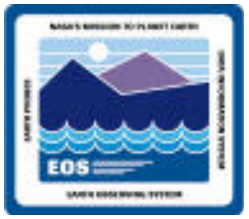
Step 4: Develop the Application Main

Managed Server Process

Development Example

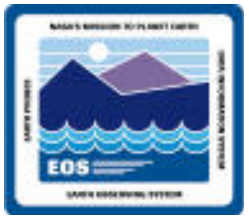


MyManagedServerProc Class Definition




```
Class MyManagedServerProc: public EcPfManagedServer
{
     // My Attributes
      ...
     // Constructor, Destructor, operations
     ....
     Start(...);           // One development option
     // Overriding methods
     PfShutdown(...);      // Mandatory
     PfGetShutdownSec(...); // Special Case
     ...
}
```

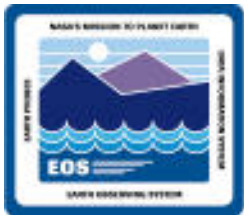
Class MyManagedServerProc Development



Constructor

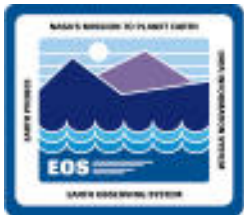
```
MyManagedServerProc::MyManagedServerProc(int *argc, char **argv)
                    : EcPfManagedServer(argc, argv)
{
        // Application related programming
        // Process application-specific command line arguments
        ...
        // Other setup functions
        ...
        // Get process data
        PerhapsIneedThis = PfGetMode();
        PerhapsIneedAlsoThis = PfGetProfileName();
        ...
        // Set my preferred protocol
        PfSetProtocolPolicy(protocol);
        ...

}
```

Main Program for PF




```
//extern ecsserver
MyManagedServerProc*    ecsServer;

int main (int argc, char** argv)
{
  ……
//Instantiates the MyManagedServerProc, this will in turn execute //
constructors of EcPfManagedServer, EcPfGenServer and EcPfGenProcess. //
Command line arguments are being passed, which will be used later by //
EcPfGenServer and EcPfGenProcess to obtain class attribute values.
  ecsServer  = new MyManagedServerProc (argc, argv, &st);
// check return status here
// Call PfInit to perform  initialization.
  PfInit ();
  Start ();
  .......
  }
```

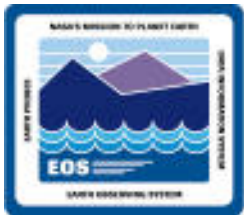
Start Up Drill Down



Construct EcPfManagedServerObject

```
// This constructor will execute the constructor of EcPfGenServer,
// which in turn will the execute constructor of EcPfGenProcess.

EcPfManagedServer::EcPfManagedServer(int argc, char** argv,
                                     EcUtStatus st)
                  :EcPfGenServer(argc, argv, st)
{
// Initialize the pointer to the EcAgManager object
// Check return status of EcPfGenServer Constructor
// Any application specific set up, such as parsing application specific
// command line arguments, calling Get/Set methods, and other set up
// process can be done here.
}
```

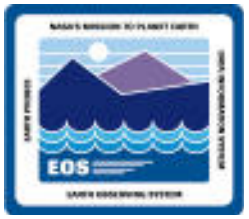
Start Up Drill Down



Read the config file and parse the command line arg

```
// This constructor calls the constructor for the base class DCEServer, so
// that the global object pointer theServer gets set to point to the Process
// Framework Server Object (ecsServer) when it is constructed.
  EcPfGenServer::EcPfGenServer(int argc, char** argv, EcUtStatus st)
                          : EcPfGenProcess(argc, argv,st), DCEServer()
{
// Initialize class attributes
// Private function PfSetAttrFromConfigFile() is called to read class //
attribute values from Configuration File.  Name of the Configuration File //
is provided from the command line arguments.
  status = PfSetAttrFromConfigFile();
// Call PfSetAttrFromArgv(argc, argv) function to parse the command line //
arguments and overwrite, or set class attributes.
  status = PfSetAttrFromArgv(argc, argv);
  .......
  }
```

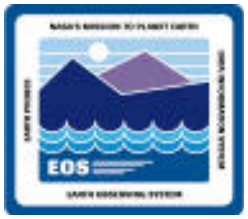
Start Up Drill Down



PfInit()

```
// PfInit is called by application main(), to initialize the managed server
// process.  In this method, the initialization method of EcPfGenServer
// (PfGenServerInit()) is invoked.  The EcAgManager object is instantiated,
// and registered with GSO.
EcUtStatus EcPfManagedServer::PfInit()
{
  ....
// call PfGenServerInit(). It will do any combination of the following:
// - DCE related initialization;
// - Initializes Message Passing and SRF
// - Initializes FTP process
 status = PfGenServerInit();
// Obtain execution name, application ID, and program ID from Get methods
// of EcPfGenProcess class.  These values will be used to construct
// EcAgManager object later
 execution_name = PfGetExecName();
   AppID = PfGetAppID();
   ProgID = PfGetProgramID();
 // Instantiates  EcAgManager object.  EcAgManager is an distributed object.
 // ECS application uses the server side of EcAgManager to communicate with
 // the MSS subagent.
   myMSSMgrPtr = new EcAgManager(execution_name,objUuid, AppID, ProgID):
....
}
```

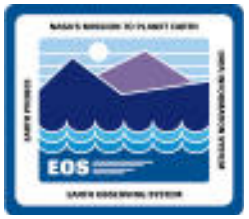
Start Up Drill Down



Start()

```
// This method will be provided by the application.  This method
// will contain whatever needs to set up the server application process.
EcUtStatus MyManagedServerProc::Start()
{
// Create server manager objects
   sleeper_1_0_Mgr   SleeperObj(objUuid);
// If security is used, check and create ACLs and ACL databases
  ....
   DCEAclSchema *theSchema = EcXSeSecurity->CreateAclSchema(status, 8)
  …
  status = PfRegisterObject(SleeperObj, true); //Register objects with GSO
  …
status = PfRegisterMetric(MgmtLevel,MetricObjPtr);//Register Metric with MSS
  …
// If needed, create FTP and Message Passing processes here
  …
//Finally, tell MSS to start monitoring this process and start listening
status = PfStart(); // This must be the last instruction
return status;
}
```

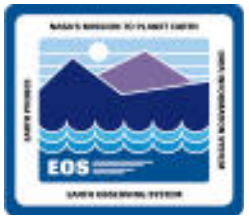
Shutdown Drill Down



PfShutdown(level)

```
EcUtStatus EcPfManagedServer::PfShutdown(EcTAgMgmtLevel  ShutdownLevel,
                                         Int  ShutdownReason,
                                         Int  gracefulflag)
{
  // This method will be called by PF upon receiving the shutdown request
 // This method will do whatever needs to be done to shutdown the
// application gracefully, such as notify client, log off databases,etc.
// PF will take care of DCE cleanup.

// If the application does not shutdown within the indicated time frame,
// (indicated from PfGetShutdownSec()), the operator will be given an
//opportunity to kill the application or let it continue to operate.
   ...
}
```

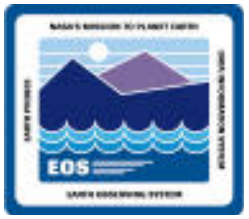
Shutdown Drill Down



PfGetShutdownSec()

```
// This method needs to be overridden by the application to
// provide an estimation of shutdown time the application needs.
// If this implementation is not provided, a default number of second
// will be provided by the process framework.

int MyManagedServerProc::PfGetShutdownSec(EcTAgMgmtLevel level)
{
   ....
   int numOfSecs;
// Calculate estimated shutdown second
   numOfSecs = MyCalculateShutdown();
   ....
  return (numOfSecs);
}
```

Shutdown Drill Down



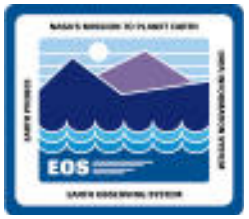
PfExecShutdown(level)

```
// This method is called by the MSS sub-agent requesting a shutdown
// be performed.  This method in turn calls theServer->Shutdown() to
// shutdown the server process.

EcUtStatus EcPfManagedServer::PfExecShutdown(EcTAgMgmtLevel level
                                             int  gracefulFlag)
{
   ....
   Shutdown();

// theServer->Shutdown(), at this point, server will
// return from listen, PfShutdown() will be called
// to perform application specific shutdown.
   ...
}
```

Event Logging Code

```
Widget_1_0_Mgr::Widget()
{
    ...
    // At this point I decide to log an event (option 1)
    theEvent = new EcAgEvent( p1, p2, p3, p4 );
    ecsServer->PfProcessEvent( theEvent );

    // New implementation (Option 2)
    theEvent = new EcLgErrorMsg( error, class, severity );
    ecsServer->PfProcessErrorMsg( theEvent );
    ...
}
```

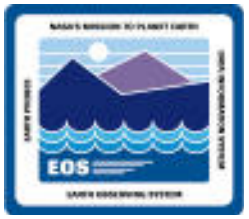
Server Application (Code)
Other examples



Getting Values

```
...

// At this point I need some values
myVariable = PfGetMode();
...
myVariable = PfGetPID();
...
myVariable = PfGetAppID();
...
myVariable = PfGetMajorVersion();
...
```

PF Design Issues



1. Configuration File

Work-off plan

- Revisit configuration file options
- Coordinate with Rel A development
- Target date for completion 06/01/96

2. SRF Client Integration

Work-off plan

- Better understand the needs of SRF client
- Specialize a new lightweight server class from the Generic Server Class
- Target date for completion 06/01/96